Evolving Multi Rover Systems in Dynamic and Noisy Environments

Kagan Tumer[1] and Adrian Agogino[2]

[1] NASA Ames Research Center
Mailstop 269-4
Moffett Field, CA 94035, USA
ktumer@mail.arc.nasa.gov
[2] Adrian Agogino
UC Santa Cruz, NASA Ames Research Center
Mailstop 269-3
adrian@email.arc.nasa.gov

In this chapter, we address how to evolve control strategies for a collective: a set of entities that collectively strives to maximize a global evaluation function that rates the performance of the full system. Addressing such problems by directly applying a global evolutionary algorithm to a population of collectives is unworkable because the search space is prohibitively large. Instead, we focus on evolving control policies for each member of the collective, where each member is trying to maximize the fitness of its own population. The main difficulty with this approach is creating fitness evaluation functions for the members of the collective that induce the collective to achieve high performance with respect to the system level goal. To overcome this difficulty, we derive member evalution functions that are both aligned with the global evaluation function (ensuring that members trying to achieve high fitness results in a collective with high fitness) and sensitive to the fitness of each member (a member's fitness depends more on its own actions than on actions of other members).

In a difficult rover coordination problem in dynamic and noisy environments, we show how to construct evaluation functions that lead to good collective behavior. The control policy evolved using aligned and member-sensitive evaluations outperforms global evaluation methods by up to a factor of four. In addition we show that the collective continues to perform well in the presence of high noise levels and when the environment is highly dynamic. More notably, in the presence of a larger number of rovers or rovers with noisy sensors, the improvements due to the proposed method become significantly more pronounced.

1 Introduction

In this chapter we show how to extend evolutionary control methods to dynamic domains that contain many devices that need to be controlled in the presence of noise. These methods are applicable to many distributed domains such as coordinating multiple robots, controlling constellations of satellites, and routing over a data network promises significant application opportunities [3, 11, 14]. In this chapter we specifically look at the problem of coordinating multiple rovers in such a way that they maximize their collective observational capability.

The challenge in this problem is efficiently evolving control policies for the rovers, such that they collectively maximize a single objective function, which is in general a non-linear function over the actions of all the rovers. The straight-forward approach to this problem is to directly evolve the entire collective by using a population of collectives and having the evolutionary operators work directly on the collective to produce a solution with high global fitness. Unfortunately this method is impractical at best and impossible at worst, since the search space for such an approach is simply too large for all but the simplest problems. Instead we will approach this problem by having each rover in the collective have its own population of control policies, using its own fitness evaluation function to evaluation these control policies. The key issue in such an approach is to ensure that the rover fitness evaluation function possesses the following two properties: (i) it is aligned with the global evaluation function, ensuring that the rovers that maximize their own fitness do not hinder one another and hurt the fitness of the collective; and (ii) it is sensitive to the fitness of the rover, ensuring that it provides the right selective pressure on the rover (i.e., it limits the impact of other rovers in the fitness evaluation function).

In this chapter we show how to create fitness evaluation functions that have these properties. We then show how to use them in a multi-rover domain that is challenging in the following two ways:

1. The environment is dynamic, meaning that the conditions under which the rovers evolve changes with time. The rovers need to evolve general control policies, rather than specific policies tuned to their current environment.
2. The rovers' sensors and actuators are noisy, meaning that the signals they receive from the environment are not reliable and the control signals that the robot sends out are not reliably carried out. The rovers need to demonstrate that the control policies are not sensitive to such fluctuations in sensor readings and control outputs.

This domain is modeled reflect the important properties evolutionary control systems need to have to be deployed. Significantly this domain does not have any "episodes" or "trials." The environment changes continuously and the rovers move continuously where neither the environment or rovers' positions are ever reset. The rovers must evolve in-situ and be able to use the control policies in an environment different from what they were evolved in.

In Section 2 we discuss the properties needed in a collective, how to evolve rovers using evaluation functions possessing such properties along with a discussion of related work. In section 3 we present the "Rover Problem" where a planetary rovers in a collective use neural networks to determine their movements based on a continuous-valued array of sensor inputs. Section 4 presents the performance of the rover collective evolved using rover evaluation functions in dynamic, noisy and communication limited domains. The results show the effectiveness of the rovers in gathering information is 400% higher with properly derived rover fitness functions than in rovers using a global evaluation function. Finally Section 5 we discuss the implication of these results and their applicability to different domains.

2 Evolving a Collective

In general, one has three possible approaches based on evolutionary computation to design control policies for collectives.

1. One can operate directly on the collective, treating it as an instance of a solution and operate on populations of collectives. In this case, the standard evolutionary algorithms are used to select for the collective that best satisfies a predetermined global evaluation function.
2. One can operate on members in the collective, treating each rover as an instance of a solution and operate of populations of rovers. In this case, the evolutionary algorithms are used to select the rovers constituting the collective based on how a given rover satisfies *the predetermined global evaluation function.*
3. One can operate on members in the collective, treating each rover as an instance of a solution and operate of populations of rovers. In this case, the evolutionary algorithms are used to select the rovers constituting the collective based on how a given rover satisfies *a specialized rover evaluation function tuned to the fitness of that rover.*

The first method presents a computationally daunting task in all but the simplest problems. Finding good control strategies is difficult enough for single controllers, but the search space become prohibitively large when they are concatenated into an "individual" representing the full collectives. Even if good rovers are present in the collective, there is no mechanism for isolating and selecting them when the collective to which they belong performs poorly. As a consequence, this approach is practically unworkable in large continuous domains.

The second method addresses part of the issue: Because the rovers in the collective are evolved independently, it avoids the explosion of the state space. However, this method introduces a new problem: How is a rover's evolution guided when the evaluation function depends on the fitness of all the other rovers? In small collectives, this method provides good solutions, but as the collectives size increases, this problem becomes more and more acute. As a consequence, this approach, though preferable to the first approach in some ways, is unlikely to provide good solutions in large collectives.

The third method provides a specialized rover evaluation function for each rover. This approach, cleans up the fitness signal a rover receives, but introduces a new twist to the problem: How does one ensure that the specialized rover evaluation functions are aligned with the global evaluation function? In other words, the fundamental question is how to guarantee that the collective evolved using rover evaluation functions will have a high fitness with respect to the global evaluation function. In this chapter we discuss the second and third approaches, focusing on how to select rover evaluation function in a formal manner as discussed below.

2.1 Rover Evaluation Function Properties

Given a global evaluation function to be maximized, this section presents the desirable properties that rover-specific evaluation functions must have. Let the **global evaluation function** be given by $G(z)$, where z is the state of the full system (e.g., the position of all the rovers in the system, along with their relevant internal parameters and the state of the environment). Let the **rover evaluation function** for rover i be given by $g_i(z)$. The first desirable proberty we want the private evaluation functions of each agent to have is high *factoredness* with respect to G, intuitively meaning that an action taken by an agent that improves its private evaluation function also improves the global evaluation function (i.e. G and g_η are aligned). Formally, the degree of factoredness between g_i and G is given by:

$$\mathcal{F}_{g_i} = \frac{\int_z \int_{z'} u[(g_i(z) - g_i(z'))\ (G(z) - G(z'))]dz'dz}{\int_z \int_{z'} dz'dz} \tag{1}$$

where z' is a state which only differs from z in the state of rover i, and $u[x]$ is the unit step function, equal to 1 when $x > 0$. Intuitively, a high degree of factoredness between g_i and G means that a rover evolved to maximize g_i will also maximize G.

Second, the rover evaluation function must be more sensitive to changes in that rover's fitness than to changes in the fitness of other rovers in the collective. Formally we quantify the *rover-sensitivity* of evaluation function g_i, at z as:

$$\lambda_{i,g_i}(z) = E_{z'}\left[\frac{\|g_i(z) - g_i(z - z_i + z'_i)\|}{\|g_i(z) - g_i(z' - z'_i + z_i)\|}\right] \tag{2}$$

where $E_{z'}[\cdot]$ provides the expected value over possible values of z', and $(z - z_i + z'_i)$ notation specifies the state vector where the components of rover i have been removed from state z and replaced by the components of rover

i from state z'. So at a given state z, the higher the rover-sensitivity, the more $g_i(z)$ depends on changes to the state of rover i, i.e., the better the associated signal-to-noise ratio for i. Intuitively then, higher rover-sensitivity means there is "cleaner" (e.g., less noisy) selective pressure on rover i.

As an example, consider the case where the rover evaluation function of each rover is set to the global evaluation function, meaning that each rover is evaluated based on the fitness of the full collective (e.g., approach 2 discussed in Section 2). Such a system will be fully factored by the definition of Equation 1. However, the rover fitness functions will have low rover-sensitivity (the fitness of each rover depends on the fitness of all other rovers).

2.2 Difference Evaluation Functions

Let us now focus on improving the rover-sensitivity of the evaluation functions. To that end, consider **difference** evaluation functions [17], which are of the form:

$$D_i \equiv G(z) - G(z_{-i} + c_i) \qquad (3)$$

where z_{-i} contains all the states on which rover i has no effect, and c_i is a fixed vector. In other words, all the components of z that are affected by rover i are replaced with the fixed vector c_i. Such difference evaluation functions are fully factored no matter what the choice of c_i, because the second term does not depend on i's states [17] (e.g., D and G will have the same derivative with respect to z_i). Furthermore, they usually have far better rover-sensitivity than does a global evaluation function, because the second term of D removes some of the effect of other rovers (i.e., noise) from i's evaluation function. In many situations it is possible to use a c_i that is equivalent to taking rover i out of the system. Intuitively this causes the second term of the difference evaluation function to evaluate the fitness of the system without i and therefore D evaluates the rover's contribution to the global evaluation.

Though for linear evaluation functions D_i simply cancels out the effect of other rovers in computing rover i's evaluation function, its applicability is not restricted to such functions. In fact, it can be applied to any linear or non-linear global utility function. However, its effectiveness is dependent on the domain and the interaction among the rover evaluation functions. At best, it fully cancels the effect of all other rovers. At worst, it reduces to the global evaluation function, unable to remove any terms (e.g., when z_{-i} is empty, meaning that rover i effects all states). In most real world applications, it falls somewhere in between, and has been successfully used in many domains including rover coordination, satellite control, data routing, job scheduling and congestion games [3, 15, 17]. Also note that the computation of D_i is a "virtual" operation in that rover i computes the impact of its not being in the system. There is no need to re-evolve the system for each rover to compute its D_i, and computationally it is often easier to compute than the global evaluation function [15]. Indeed in the problem presented in this chapter, for rover i, D_i is easier to compute than G is (see details in Section 4).

2.3 Related Work

Evolutionary computation has a long history of success in single agent and multi-agent control problems [16, 10, 7, 2, 1]. Advances in evolutionary computation methods in single agent domains tend to result from improvements in search methods. In [10] this is accomplished through fuzzy rules in a helicopter control problem, while in [16] cellular encoding is used to improve performance on pole-balancing control. Similarly [7] addresses planetary rover control by having genetic algorithms search through a space of plans generated from a planning algorithm.

Many advances in evolutionary computation for multi-agent control have been accomplished through the use of domain specific fitness functions. Ant colony algorithms [6] solve the coordination problem by utilizing "ant trails" that provide implicit fitness functions resulting in good performance in path-finding domains. In [2], the algorithm takes advantage of a large number of agents to speed up the evolution process in certain domains, but uses greedy fitness functions that are not generally factored. Also outside of evolutionary computation, coordination between a set of mobile robots has been accomplished through the use of hand-tailored rewards designed to prevent greedy behavior [12]. While highly successful in many domains all of these methods differ from the methods used in this chapter in that they lack a general framework for efficient evolution in multi-agent systems.

3 Continuous Rover Problem

In this section, we show how evolutionary computation with the difference evaluation function can be used effectively in the Rover Problem[3]. In this problem, there is a collective of rovers on a two dimensional plane, which is trying to observe points of interests (POIs). Each POI has a value associated with it and each observation of a POI yields an observation value inversely related to the distance the rover is from the POI. In this chapter the distance metric will be the squared Euclidean norm, bounded by a minimum observation distance, δ_{min}:[4]

$$\delta(x,y) = min\{\|x-y\|^2, \delta_{min}^2\} \,. \tag{4}$$

[3] This problem was first presented in [3].

[4] The square Euclidean norm is appropriate for many natural phenomenon, such as light and signal attenuation. However any other type of distance metric could also be used as required by the problem domain. The minimum distance is included to prevent singularities when a rover is very close to a POI.

The global evaluation function is given by:

$$G = \sum_t \sum_j \frac{V_j}{\min_i \delta(L_j, L_{i,t})} , \tag{5}$$

where V_j is the value of POI j, L_j is the location of POI j and $L_{i,t}$ is the location of rover i at time t. Intuitively, while any rover can observe any POI, as far as the global evaluation function is concerned, only the closest observation matters[5].

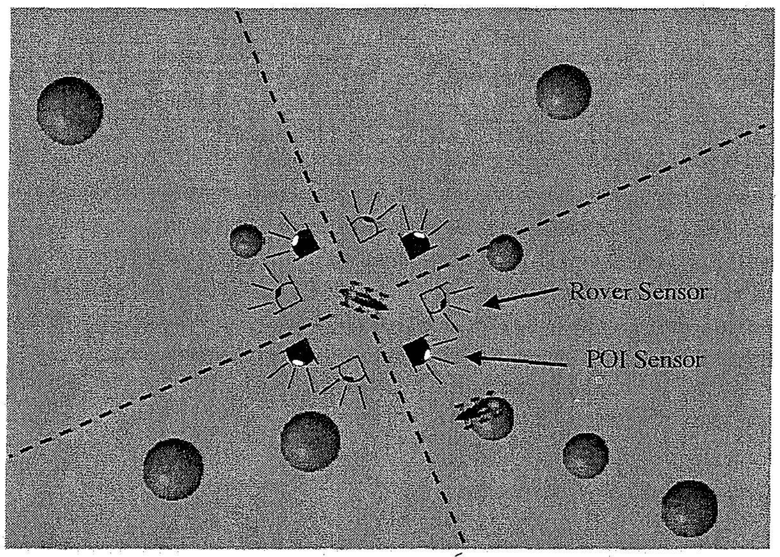

Fig. 1. Diagram of a Rover's Sensor Inputs. The world is broken up into four quadrants relative to rover's position. In each quadrant one sensor senses points of interests, while the other sensor senses other rovers.

3.1 Rover Capabilities

At every time step, the rovers sense the world through eight continuous sensors. From a rover's point of view, the world is divided up into four quadrants relative to the rover's orientation, with two sensors per quadrant (see Figure 1). For each quadrant, the first sensor returns a function of the POIs in the quadrant at time t. Specifically the first sensor for quadrant q returns the sum

[5] Similar evaluation functions could also be made where there are many different levels of information gain depending on the position of the rover. For example 3-D imaging may utilize different images of the same object, taken by two different rovers.

of the values of the POIs in its quadrant divided by their squared distance to the rover and scaled by the angle between the POI and the center of the quadrant:

$$s_{1,q,j,t} = \sum_{j \in J_q} \frac{V_j}{\delta(L_j, L_{i,t})} \left(1 - \frac{|\theta_{j,q}|}{90}\right) \tag{6}$$

where J_q is the set of observable POIs in quadrant q and $|\theta_{j,q}|$ is the magnitude of the angle between POI j and the center of the quadrant. The second sensor returns the sum of square distances from a rover to all the other rovers in the quadrant at time t scaled by the angle:

$$s_{2,q,i,t} = \sum_{i' \in N_q} \frac{1}{\delta(L_{i'}, L_{i,t})} \left(1 - \frac{|\theta_{i',q}|}{90}\right) \tag{7}$$

where N_q is the set of rovers in quadrant q and $|\theta_{i',q}|$ is the magnitude of the angle between rover i' and the center of the quadrant.

The sensor space is broken down into four regions to facilitate the input-output mapping. There is a trade-off between the granularity of the regions and the dimensionality of the input space. In some domains the tradeoffs may be such that it is preferable to have more or fewer than four sensor regions. Also, even though this chapter assumes that there are actually two sensors present in each region at all times, in real problems there may be only two sensors on the rover, and they do a sensor sweep at 90 degree increments at the beginning of every time step.

3.2 Rover Control Strategies

With four quadrants and two sensors per quadrant, there are a total of eight continuous inputs. This eight dimensional sensor vector constitutes the state space for a rover. At each time step the rover uses its state to compute a two dimensional output. This output represents the x, y movement relative to the rover's location and orientation. Figure 2 displays the orientation of a rover's output space.

The mapping from rover state to rover output is done through a Multi Layer Perceptron (MLP), with eight input units, ten hidden units and two output units [6]. The MLP uses a sigmoid activation function, therefore the outputs are limited to the range $(0, 1)$. The actual rover motions dx and dy, are determined by normalizing and scaling the MLP output by the maximum distance the rover can move in one time step. More precisely, we have:

$$dx = d_{max}(o_1 - 0.5)$$
$$dy = d_{max}(o_2 - 0.5)$$

[6] Note that other forms of continuous reinforcement learners could also be used instead of evolutionary neural networks. However neural networks are ideal for this domain given the continuous inputs and bounded continuous outputs.

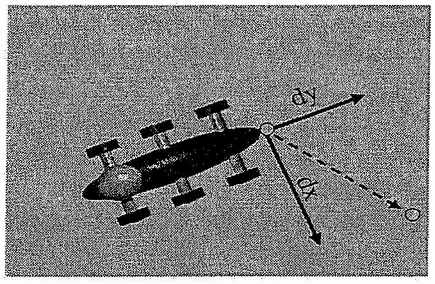

Fig. 2. Diagram of a Rover's Movement. At each time step the rover has two continuous outputs (dx, dy) giving the magnitude of the motion in a two directional plane relative to the rover's orientation.

where d_{max} is the maximum distance the rover can move in one time step, o_1 is the value of the first output unit, and o_2 is the value of the second output unit.

3.3 Rover Selection

The MLP for a rover is selected using an evolutionary algorithm as highlighted in approaches two and three in Section 2. In this case, each rover has a population of MLPs. At each N time steps (N set to 15 in these experiments), the rover uses ϵ-greedy selection ($\epsilon = 0.1$) to determine which MLP it will use (e.g., it it selects the best MLP from its population with 90% probability and a random MLP from its population with 10% probability). The selected MLP is then mutated by adding a value sampled from the Cauchy Distribution (with scale parameter equal to 0.3) to each weight, and is used for those N steps. At the end of those N steps, the MLP's performance is evaluated by the rover's evaluation function and re-inserted into its population of MLPs, at which time, the poorest performing member of the population is deleted. Both the global evaluation for system performance and rover evaluation for MLP selection is computed using an N-step window, meaning that the rovers only receive an evaluation after N steps.

While this is not a sophisticated evolutionary algorithm, it is ideal in this work since our purpose is to demonstrate the impact of principled evaluation functions selection on the performance of a collective. Even so, this algorithm has shown to be effective if the evaluation function used by the rovers is factored with G and has high rover-sensitivity. We expect more advanced evolutionary computation algorithms used in conjunction with these same evaluation functions to improve the performance of the collective further.

3.4 Evolving Control Strategies in a Collective

The key to success in this approach is to determine the correct rover evaluation functions. In this work we test three different evaluation function for rover selection. The first evaluation function is the global evaluation function (G), which when implemented results in approach two discussed in Section 2:

$$G = \sum_t \sum_j \frac{V_j}{\min_i \delta(L_j, L_{i,t})} \tag{8}$$

The second evaluation function is the "perfectly rover-sensitive" evaluation function (P):

$$P_i = \sum_t \sum_j \frac{V_j}{\delta(L_j, L_{i,t})} \tag{9}$$

The P evaluation function is equivalent to the global evaluation function in the single rover problem. In a collective of rover setting, it has infinite rover-sensitivity (in the way rover sensitivity is defined in Section 2). This is because the P evaluation function for a rover is not affected by the states of the other rovers, and thus the denominator of Equation 2 is zero. However the P evaluation function is not factored. Intuitively P and G offer opposite benefits, since G is by definition factored, but has poor rover-sensitivity. The final evaluation function is the difference evaluation function. It does not have as high rover-sensitivity as P, but is still factored like G. For the rover problem, the difference evaluation function, D, becomes:

$$D_i = \sum_t \left[\sum_j \frac{V_j}{\min_{i'} \delta(L_j, L_{i',t})} - \sum_j \frac{V_j}{\min_{i' \neq i} \delta(L_j, L_{i,t})} \right] .$$

The second term of the D is equal to the value of all the information collected if rover i were not in the system. Note that for all time steps where i is not the closest rover to any POI, the subtraction leaves zero. As mentioned in Section 2.2, the difference evaluation in this case is easier to compute as long as rover i knows the position and distance of the closest rover to each POI it can see. In that regard it requires knowledge about the position of fewer rovers than if it were to use the global evaluation function.

4 Results

We performed extensive simulation to test the effectiveness of the different rover evaluation function under a wide variety of environmental conditions and rover capabilities. In these experiments, each rover had a population of MLPs of size 10. The world was 75 units long and 75 units wide. All of the rovers started the experiment at the center of the world. Unless otherwise state as in the scaling experiments, there were 30 rovers in the simulations. The maximum distance the rovers could move in one direction during a time step, d_{max}, was set to 3. The rovers could not move beyond the bounds of the world. The minimum observation distance, δ_{min}, was equal to 5.

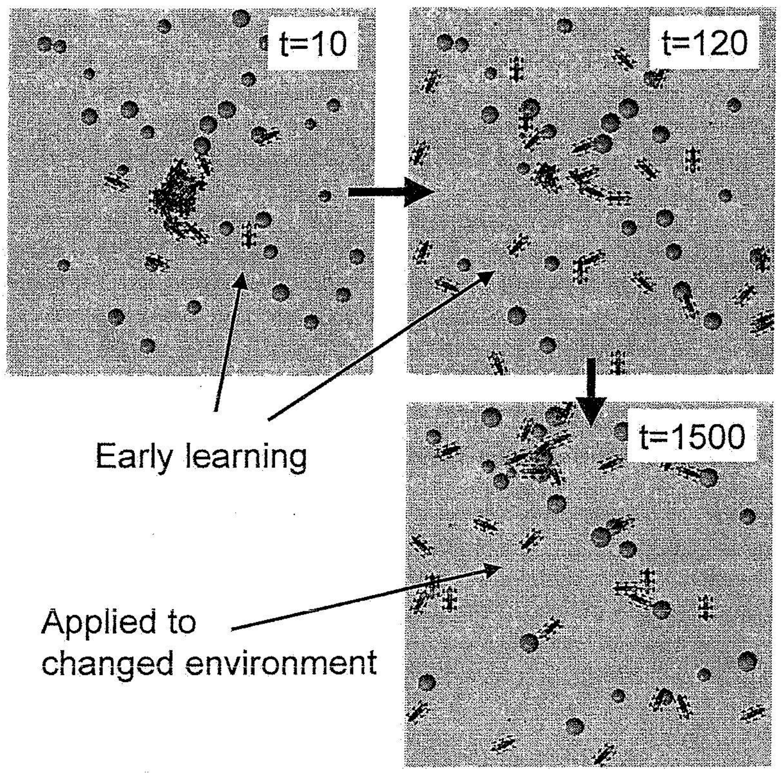

Fig. 3. Sample POI Placement. Left-Top: Environment at time = 10. Right-Top: Environment at time = 120. Bottom: Environment at time = 1500. Environment at time step 10 is similar to environment at time step 120, but significantly different than environment at time step 1500. Rovers must to able to use their control policies evolved from earlier time step, in future changed environments.

In these experiments the environment was dynamic, meaning that the POI locations and values changed with time. There were as many POIs as rovers, and the value of each POI was set to between three and five using a uniformly random distribution. In these experiments, each POI disappeared with probability 2.5%, and another one appeared with the same probability at 15 time step intervals. Because the experiments were run for 3000 time steps, the initial and final environments had little similarities. All results were averaged over at least one hundred independent trials (except for the seventy agent runs where there were thirty trials). For each experiment and trial the weights of the neural network were set to random using the Cauchy distribution (parameter of 0.5).

Results for episodic environments where the agents were restored to their initial state at the end of each trial were reported in [3]. In such a case the rovers evolve specific control policies tuned to the particular environment in which they are trained. Though useful in domains where the simulated environment closely matches the environment in which the rovers will operate, this approach has limited applicability in general. A more desirable approach is for the rovers to evolve efficient policies that are solely based on their sensor inputs and not on the specific configuration of the POIs. The dynamic environment experiments reported here explore this premise and provide rover control policies that can be generalized from one set of POIs to another, regardless of how significantly the environment changes. Figures 3 shows an instance of change in the environment throughout a simulation. The final POI set is not particularly close to the initial POI set and the rovers are forced to focus on the sensor input-output mappings rather than focus on regions in the (x, y) plane.

4.1 Evolution in Noise Free Environment

The first set of experiments tested the performance of the three evaluation functions in a dynamic noise-free environment for 30 rovers. Figure 4 shows the performance of each evaluation function. In all cases, performance is measured by the same global evaluation function, regardless of the evaluation function used to evolve the system. All three evaluation functions performed adequately in this instance, though D_i outperformed both P and G.

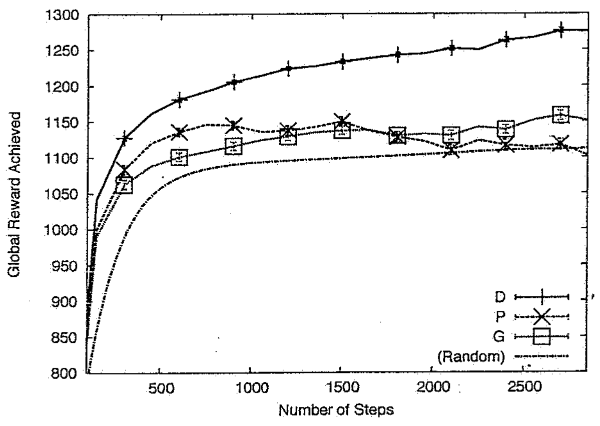

Fig. 4. Performance of a 30-rover collective for all three evaluation functions in noise-free environment. Difference evaluation function provides the best collective performance because it is both factored and rover-sensitive.

The evolution of this system also demonstrate the different properties of the rover evaluation functions. After initial improvements, the system with the G evaluation function improves slowly. This is because the G evaluation function has low rover-sensitivity. Because the fitness of each rover depends on the state of all other rovers, the noise in the system overwhelms the evaluation function. P on the other hand has a different problem: After an initial improvement, the performance of systems with this evaluation function decline. This is because though P has high rover-selectivity, it is not fully factored with the global evaluation function. This means that rovers selected to improve P do not necessarily improve G. D on the other hand is both factored and has high rover-sensitivity. As a consequence, it continues to improve well into the simulation as the fitness signal the rovers receive are not swamped by the states of other rovers in the system. This simulation highlights the need for having evaluation function that are both factored with the global evaluation function and have high rover-sensitivity. Having one or the other is not sufficient.

4.2 Scaling in Noise-free Environments

The second set of experiments focuses on the scaling properties of the three evaluation functions in a dynamic noise-free environment. Figure 5 shows the performance of each evaluation function at t=3000 for a collective of 10 to 70 rovers (where the number of POIs is equal to the number of rovers). For each case, the results are qualitatively similar to those reported above, except where there are only 5 rovers, in which case P performs as well as G. This is not surprising since with so few rovers, there are almost no interactions among the rovers, and in as large a space as the one used here, the 5 rovers act almost independently.

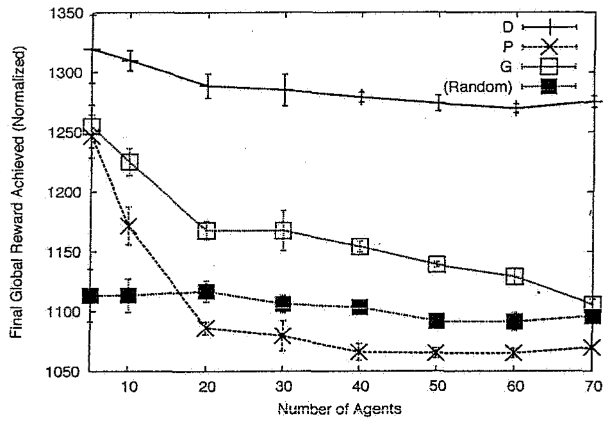

Fig. 5. Scaling properties of the three evaluation functions. The D evaluation function not only outperforms the alternatives, but the margin by which it outperforms them increases as the size of the collective goes up.

As the size of the collective increases though, an interesting pattern emerges: The performance of both P and G drop at a faster rate than that of D. Again, this is because G has low rover-sensitivity and thus the problem becomes more pronounced as the number of rovers increases. Similarly, as the number of rovers increases, P becomes less and less factored. In fact the performance of rovers using P is even worse than random when there are many rovers because the rovers' greedy actions make them focus on only a few POIs, while the random rovers at least distribute themselves among the POIs. D on the other hand handles the increasing number of rovers quite effectively. Because the second term in Equation 3 removes the impact of other rovers from rover i, increasing the number of rovers does very little to limit the effectiveness of this rover evaluation function. This is a powerful result suggesting that D is well suited to evolve large collectives in this and similar domains where the interaction among the rovers prevents both G and P from performing well. This result also supports the intuition expressed in Section 2 that approach 2 (i.e., evolving rovers based on the fitness of the full collective) is ill-suited to evolving collectives in all but the smallest examples.

4.3 Evolution in Noisy Environment

The third set of experiments tested the performance of the three evaluation functions in a dynamic environment for 30 rovers with noisy sensors. Figure 6 shows the performance of each evaluation function when both the input sensors and the output values of the rovers have 5% noise added. All three evaluation functions handle the noise well. This result is encouraging in that it shows that not only simple evaluation functions such as P can handle moderate amounts of noise in their sensors and outputs, but so can D. In other words, considering the impact of other rovers to yield a factored evaluation function does not cause to compound moderate noise in the system and overwhelm the rover evaluation.

Figure 7 shows the noise sensitivity of the three different evaluation functions. The performance is reported as a function of additive noise to sensors as the percentage shown on the x-axis (e.g., 0.5 means the magnitude of the added noise is half that of the sensor value.) The results are shown as the D is the most sensitive to high levels of noise, though even at 80% noise it still far outperforms both G and P. This is an encouraging result in the power of the D evaluation function is that it "cleans up" the evaluation function for a rover (e.g., it has high rover-sensitivity). Adding noise, starts to cancel this property of D, but even when half the signal being noise does not prevent D from far outperforming D and P. Interestingly, rovers using P actually perform marginally better as noise increases, demonstrating the importance of factoredness. Adding noise to the system actually hindered these rovers from learning some counter-productive actions.

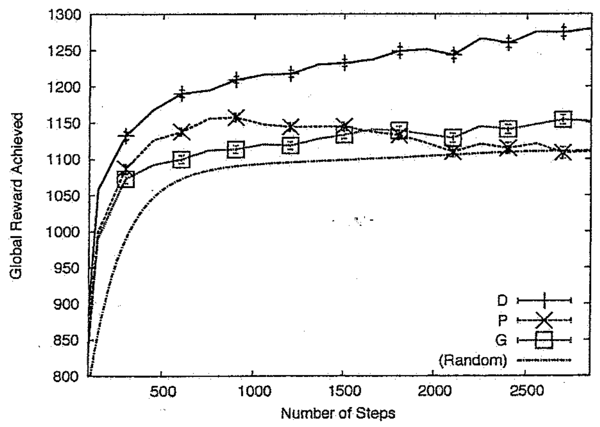

Fig. 6. Performance of a 30-rover collective for all three evaluation functions when the rover sensors and outputs have 5% noise.

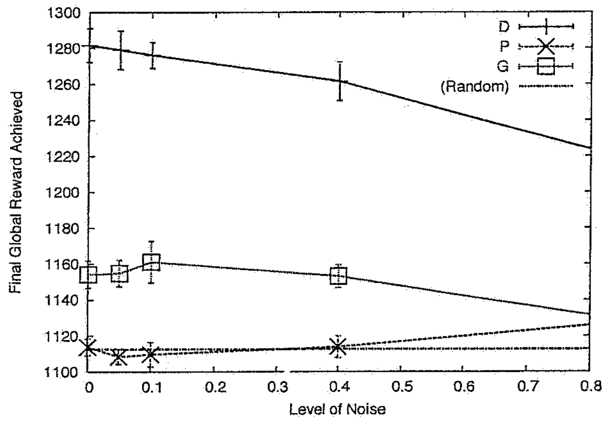

Fig. 7. Sensitivity of the three evaluation functions to the degree of noise in their sensors.

5 Discussion

Extending the success of evolutionary algorithms in continuous single-controler domains to large, distributed multi-controller domains has been a challenging endeavor. Unfortunately the direct approach of having a population of collectives and applying the evolutionary algorithm to that population results in a prohibitively large search space in most cases. As an alternative, this chapter presents a method for providing rover specific evaluation functions to directly evolve individual rovers in collective. The fundamental issue in this approach is in determining the rover specific evaluation functions that are both aligned with the global evaluation function and are as sensitive as possible to changes in the fitness of each member.

In dynamic, noise-free environments rovers using the difference evaluation function D, derived from the theory of collectives, were able to achieve high levels of performance because the evaluation function was both factored and highly rover-sensitive. These rovers performed better than rovers using the non-factored perfectly rover-sensitive evaluation and more than 400% better (over random rovers) than rovers using the hard to learn global evaluations.

We then extended these results to rovers with noisy sensors, rovers with limited communication capabilities and larger collectives. In each instance the collectives evolved using D performed better than alternative and in most cases (e.g., larger collectives) the gains due to D increase as the conditions worsened. These results show the power of using factored and rover-sensitive fitness evaluation functions, which allow evolutionary computation methods to be successfully applied to large distributed systems in real world applications where the rover sensors and actuators cannot be noise-free.

References

1. A. Agah and G. A. Bekey. A genetic algorithm-based controller for decentralized multi-agent robotic systems. In *In Proc. of the IEEE International Conference of Evolutionary Computing*, Nagoya, Japan, 1996.
2. A. Agogino, K. Stanley, and R. Miikkulainen. Online interactive neuro-evolution. *Neural Processing Letters*, 11:29–38, 2000.
3. A. Agogino and K. Tumer. Efficient evaluation functions for multi-rover systems. In *The Genetic and Evolutionary Computation Conference*, pages 1–12, Seatle, WA, June 2004.
4. T. Balch. Behavioral diversity as multiagent cooperation. In *Proc. of SPIE '99 Workshop on Multiagent Systems*, Boston, MA, 1999.
5. G. Baldassarre, S. Nolfi, and D. Parisi. Evolving mobile robots able to display collective behavior. *Artificial Life*, pages 9: 255–267, 2003.
6. M. Dorigo and L. M. Gambardella. Ant colony systems: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
7. S. Farritor and S. Dubowsky. Planning methodology for planetary robotic exploration. In *ASME Journal of Dynamic Systems, Measurement and Control*, volume 124, pages 4: 698–701, 2002.
8. D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *Proc. of Conf. on Simulation of Adaptive Behavior*, 1994.
9. F. Gomez and R. Miikkulainen. Active guidance for a finless rocket through neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Chicago, Illinois, 2003.
10. F. Hoffmann, T.-J. Koo, and O. Shakernia. Evolutionary design of a helicopter autopilot. In *Advances in Soft Computing - Engineering Design and Manufacturing, Part 3: Intelligent Control*, pages 201–214, 1999.